A simple and efficient class of functions to model arrival curve of packetised flows[*] [†]

Marc Boyer
ONERA – The French Aerospace Lab
F 31055 , Toulouse, France
marc.boyer@onera.fr

Jörn Migge
RealTime-at-Work (RTaW)
54600 Villers-lès-Nancy, France
jorn.migge@realtimeatwork.com

Nicolas Navet
INRIA / RTaW
54600 Villers-lès-Nancy, France
nicolas.navet@inria.fr

ABSTRACT

Network Calculus is a generic theory conceived to compute upper bounds on network traversal times (WCTT – Worst Case Traversal Time). This theory models traffic constraints and service contracts with *arrival curves* and *service curves*. As usual in modelling, the more realistic the model is, the more accurate the results are, however a detailed model implies large running times which may not be the best option at each stage of the design cycle. Sometimes, a trade-off must be found between result accuracy and computation time. This paper proposes a combined use of two simple class of curves in order to produce accurate results with a low computational complexity. Experiments are then conducted on a realistic AFDX case-study to benchmark the proposal against two existing approaches.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Performance attributes; G.m [**Mathematics of Computing**]: Miscellaneous—*Queueing theory*; C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

1. INTRODUCTION

Context of the problem. Aircrafts today embed hundreds of sensors, actuators and computers communicating through networks. AFDX (Avionics Full-Duplex Switched Ethernet)[1] is a widely used network technology that equips most recent large civil airplanes. Thousands of data flows, called Virtual Links (VL) in AFDX terminology, are exchanged among the nodes. To ensure the correct temporal behaviour of the applications, that are often subject to stringent timing constraints, the time between the emission and reception of each VL packet, also called the *Worst Case Traversal Time*, must be upper bounded.

Definition of the problem. An AFDX network is an Ethernet-based technology, using full duplex links. So, there are no collisions between frames, and the only indeterminism (and main cause of delays) comes from the waiting time in the switch queues. Due to the size of the system (number of VLs, number of nodes and switches, redundant networks), computationally efficient algorithms must be used, especially early in the design cycle, when design space exploration may be used for optimizing topologies, streams routing and functions allocation. But to obtain computationally efficient algorithms, some conservative (*i.e.* pessimistic) approximations have to be done in the modelling and/or the bound computation.

Existing work. If network calculus has successfully been used to compute the WCTT of the AFDX network [10, 11] and certify the A380, the computed bounds are most often larger than the real worst-cases, and significant improvements are probably not out of reach (see, for instance [2, 13] for recent works). To compute accurate bounds, a key mechanism that should be modelled is serialisation, also called grouping or shaping depending on the authors: when two flows share the same link from one systems to another, they can not reach the destination at the same time. Modelling (or ignoring) this mechanism can modify the bounds up to 40% in a realistic AFDX configuration [11, 4].

Shortcomings of existing solutions. A source of pessimism in network calculus, as it has been most often been used until yet, is the *fluid* modelling of flows. Indeed, because they have a lot of good properties ($O(n)$ complexity for the main algorithms, easy implementation, etc), concave piecewise linear function (CPL) are often used [4]. But in such models, the packet view is lost, and it leads to pessimistic approximation. Staircase functions can model the packet aspect, but not the shaping. It exists a generic class, UPP [3], which generalises both CPL and staircases, but it leads to computation times that are usually more than one order of magnitude larger than with CPL and staircase functions (see [5]).

Contribution and organisation of the paper. In this study the proposal is to make a combined use of two simple classes (staircases and CPL), and to juggle with both in the

[*](c) 2011 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the national government of France. As such, the government of France retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

[†]This work has been partially funded by French ANR agency under project id ANR-09-SEGI-009.

same algorithms, in order to not resort to more computationally involved class of functions. The paper, after a recap on network calculus (Section 2), discusses (Section 3) the problem of the flow modelling in network calculus, and the running time performance versus accuracy trade-off. Then, a common algorithms (with its proof) designed to compute bounds in FIFO algorithm is presented[1] in Section 4.1. The main contribution of the paper, the adaptation of the algorithm is presented in Section 4.2. Section 5 presents an experiment on a realistic AFDX configuration. Section 6 concludes.

2. NETWORK CALCULUS

Network calculus is a theory to obtain deterministic upper bounds in networks. It is mathematically based on the $(\wedge, +)$ dioid. This section provides a short introduction to network calculus. The reader should refer to [8, 9] for the first works, and [6, 12] for comprehensive presentations.

Basic notations. $\mathbb{N}$ denotes the set of natural numbers and $\mathbb{R}$ the set of real numbers. The minimum (resp. maximum) operator is denoted $\wedge$ (resp. $\vee$). $[x]^+ \overset{\text{def}}{=} x \vee 0$, and $\lceil x \rceil$ is the ceiling function ($\lceil x \rceil \in \mathbb{N}$, $x \leq \lceil x \rceil < x + 1$).

Network calculus mainly handles non decreasing functions, null before 0 : $\mathcal{F}$.

$$\mathcal{F} = \left\{ f : \mathbb{R} \to \mathbb{R} \,\middle|\, \begin{array}{lcl} x < y & \implies & f(x) \leq f(y) \\ x < 0 & \implies & f(x) = 0 \end{array} \right\}$$

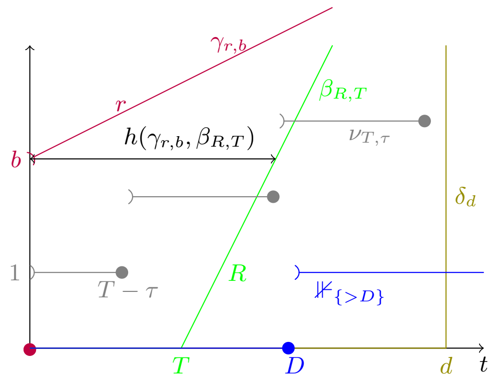

Figure 1: Common curves and illustration of the delay

There are six common curves, parametrised by real non negative values d, R, T, r, b, τ. These curves, shown in Figure 1, are: latency δ_d, rate λ_R, rate-latency $\beta_{R,T}$, token bucket $\gamma_{r,b}$, stair-case $\nu_{T,\tau}$, test $\mathbb{1}_{\{>T\}}$ and they are defined as:

$$\delta_d(t) = \begin{cases} 0 \text{ if } t \leq d \\ \infty \text{ otherwise} \end{cases} \qquad \mathbb{1}_{\{>T\}}(t) = \begin{cases} 1 \text{ if } t > 1 \\ 0 \text{ otherwise} \end{cases}$$

$$\gamma_{r,b}(t) = (rt + b)\mathbb{1}_{\{>0\}}(t) \qquad \lambda_R(t) = Rt$$

$$\beta_{R,T}(t) = R[t - T]^+ \qquad \nu_{T,\tau}(t) = \left\lceil \frac{t+\tau}{T} \right\rceil \wedge \delta_0$$

[1]It should be mentioned that, even if all the algorithm principles have been presented in the literature [11], this is the first time, to the best of our knowledge, that a complete detailed algorithm is presented and proven.

Three basic operators on $\mathcal{F}$ are of major interest; the convolution $*$, the deconvolution $\oslash$ and the sub-additive closure f^*:

$$(f * g)(t) = \inf_{0 \leq u \leq t} (f(t-u) + g(u)) \tag{1}$$

$$(f \oslash g)(t) = \sup_{0 \leq u} (f(t+u) - g(u)) \tag{2}$$

$$f^* = \delta_0 \wedge f \wedge (f * f) \wedge (f * f * f) \wedge \cdots \tag{3}$$

There are several well-known properties of theses operators and the common curves. For instance, the convolution is associative and commutative, $\delta_t * \delta_{t'} = \delta_{t+t'}$, $f * \delta_d(t) = f(t-d)$ if $t \geq d$, 0 otherwise, $f \oslash \delta_d(t) = f(t+d)$, $\beta_{R,T} = \delta_T * \lambda_R$.

Arrival and service curves. A flow is represented by its cumulative function $R \in \mathcal{F}$, where $R(t)$ is the total number of bits sent by this flow up to time t. A flow R is said to have a function α as *arrival curve* (denoted $R \prec \alpha$) iff $\forall t, s \geq 0 : R(t+s) - R(t) \leq \alpha(s)$. It means that, from any instant t, the flow R will produce at most $\alpha(s)$ new data in s time units. An equivalent condition, expressed in the $(\wedge, +)$ dioid is $R \leq R * \alpha$. This min-plus definition of arrival curves leads to three interesting results: first, if α is an arrival curve for R, also is α^*; second, if α is an arrival curve for R, any $\alpha' \geq \alpha$ also is; third, if α and α' are arrival curve for R, also is $\alpha \wedge \alpha'$.

$$R \prec \alpha \implies R \prec \alpha^* \tag{4}$$

$$R \prec \alpha, \alpha' \geq \alpha \implies R \prec \alpha^* \tag{5}$$

$$R \prec \alpha, R \prec \alpha' \implies R \prec \alpha \wedge \alpha' \tag{6}$$

A server S is a relation between input flows and some output flows, denoted $R \xrightarrow{S} R'$. A server offers a *service* of curve β (denoted $S \trianglerighteq \beta$) iff for all arrival flows, the relation between the input flow R and the output flow R', one has $R' \geq R * \beta$. In this case, $\alpha' = \alpha \oslash \beta$ is an arrival curve for R'. The delay experimented by the flow R can be bounded by the maximal horizontal difference between curves α and β, formally defined by $h(\alpha, \beta)$ (a graphical interpretation of h is shown in Figure 1).

$$h(\alpha, \beta) = \sup_{s \geq 0} \left(\inf \left\{ \tau \geq 0 \,|\, \alpha(s) \leq \beta(s + \tau) \right\} \right)$$

A server can also have a *shaping curve* $\sigma \in \mathcal{F}$, meaning that the output R' of the server always respects the shaping curve, *i.e.* $\forall t, \Delta \geq 0 : R'(t + \Delta) - R'(t) \leq \sigma(\Delta)$. It can also be said that each output has σ as arrival curve.

Sequence of servers. When a flow goes through a sequence of server S, S', then, the sequence itself can be seen as a server $S; S'$ defined by

$$R \xrightarrow{S;S'} R' \iff \exists \dot{R} : R \xrightarrow{S} \dot{R} \text{ and } \dot{R} \xrightarrow{S'} R' \tag{7}$$

These first results allow to handle linear topologies, like the one of Figure 2. Given the arrival curve α of flow R, and the services curves β, β' of network elements S, S', we are able to get a bound[2] on the delay in S: $h(\alpha, \beta)$, and another for the delay in S': $h((\alpha \oslash \beta)^*, \beta')$.

A famous result of the network calculus is known as "pay burst only once" (PBOO). It states that if a flow goes trough

[2]To effectively compute these bounds, we need algorithms computing operations like $\oslash$, $\cdot^*$, $h(\cdot, \cdot)$. This is the aim of the work published in [3] and its corresponding implementation in the COINC library [7].

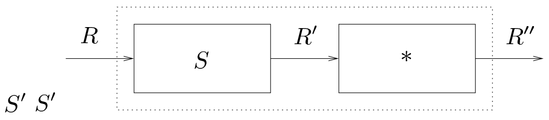

Figure 2: A flow going through two network elements in sequence

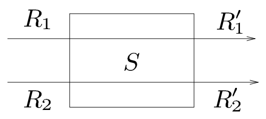

Figure 3: A single shared network element

a sequence of two servers, S and S', with respective service of curve β, β', like the one of Figure 2, then the system is identical to a flow crossing a single server $S; S'$ offering a service curve $\beta * \beta'$. The interest of PBOO comes from the fact that the end-to-end delay is smaller than the sum of local delays (*i.e.* $h(\alpha, \beta * \beta') \leq h(\alpha, \beta) + h((\alpha \oslash \beta)^*, \beta')$). And the difference can be significant in practice.

In case of more realistic topology, a network element is shared by different flows (like in Figure 3) with a given service policy (FIFO, static priority, etc). In this case, the service is offered to the aggregated flow $R = R_1 + R_2$ (*i.e.* $R_1' + R_2' \geq (R_1 + R_2) * \beta$). Note that, if each flow R_i has α_i as arrival curve, then $\alpha_1 + \alpha_2$ is an arrival curve for the aggregated flow and conversely, if α is an arrival curve for $R_1 + R_2$, it is also for each flow R_1 and R_2.

$$R_1 \prec \alpha_1, R_2 \prec \alpha_2 \implies R_1 + R_2 \prec \alpha_1 + \alpha_2 \tag{8}$$

$$R_1 + R_2 \prec \alpha \implies R_1 \prec \alpha, R_2 \prec \alpha \tag{9}$$

Recap of some needed results. Network calculus include results to compute bounds on each flow. The idea is to extract, from an *aggregated* flow, the *residual* service offered to each flow (also known as *per-flow* service). This extraction is, in a first step, local, *i.e.* on a single server. Depending on the policy, three cases arise: 1) the bounds are tight (*i.e.* there exists at least one trajectory of the system where the computed bound is reached), 2) bounds are pessimistic, or 3) it is not known whether the bounds are tight or not. The second step consists in considering a complete topology, trying to obtain an end-to-end bound better than the sum of local delays (like the PBOO result).

The classical result for the FIFO policy states that, if a FIFO server (with a service of curve β) is shared by two flows R_1, R_2 (with respective arrival curves α_1, α_2), then, for each $\theta \geq 0$, each flow R_i receives the residual service β_i^θ defined by

$$\beta_i^\theta = [\beta - \alpha_j \oslash \delta_\theta]^+ \wedge \delta_\theta \tag{10}$$

with $\{i, j\} = \{1, 2\}$ (see [12, Prop 6.4.1] for details). An equivalent definition is $\beta_i^\theta(t) = [\beta(t) - \alpha_j(t - \theta)]^+ \mathbb{1}_{\{>\theta\}}(t)$. It should be noticed that this result does not define a *single* residual service, but an *infinite set* of services, one for each value of θ. They all compute true bounds, but some are better than others. The issue is the choice of a good θ value.

Another result must be presented. It is used in [11], and states that, if d denotes the delay experimented by the aggregated flow, then a FIFO server can be under-approximated by a variable delay d, and δ_d is a residual service curve for each flow R_i. Since one knows that $h(\beta, \alpha_1 + \alpha_2)$ is an upper

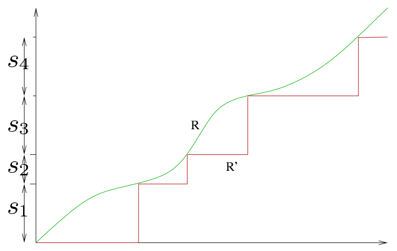

Figure 4: Illustration of the packetizer

bound of the delay, β_d is a commonly used residual delay for each R_i.

$$\beta_d = \delta_{h(\beta, \alpha_1 + \alpha_2)} \tag{11}$$

A last result of importance for this paper concerns packetizers. A packetizer P is an element that inputs some "fluid" flow R and that outputs packetized flow R'. In a more formal way, each flow R is assumed to carry a sequence of packets. Let $s_1, s_2, \ldots, s_i, \ldots$ be the size of the packets. Let $P : \mathbb{R} \to \mathbb{N}$ defined by $P(x) = \max\left\{ j \mid \sum_{i=1}^{j} s_i < x \right\}$, denotes the index of the last full packet. Then, $R'(t) = \sum_{i=1}^{P(R(t))} s_i$.

An important feature of the packetizer is that, repacking does not increase the delay. That is to say, in case of a sequence $R \xrightarrow{S} R' \xrightarrow{P} R''$, assuming that R is packetized (*i.e.* $R(t) = \sum_{i=1}^{P(R(t))} s_i$), then, the worst delay of the combined system $S; P$ is the same as for the fluid system [12, Th. 1.7.1].

$$d(R, S; P) = d(R, S) \tag{12}$$

Its does not mean that the packetizer has no effect on the flow, but when considering the worst delay, since all bits of a packet enter the combined system at the same date, since the worst delay is experience in the fluid system by the last bit, and since the last bit of a packet is not delayed by a packetizer, then, the packetizer does not add any delay.

It should be noted that, if S is a FIFO server, then $S; P$ is also a FIFO server, and eq. (11) can be used.Also, if a flow with arrival curve α crosses a packetizer, the output will have arrival curve $\alpha + l^{max}$, with l^{max} the maximal packet size of the flow.

$$R \xrightarrow{P} R', R \prec \alpha \implies R' \prec \alpha + l^{max} \tag{13}$$

Last, it must be remind that a packetized flow can have a fluid arrival curve: it is a loss of information, and lead to pessimistic over-approximation, as will be presented in next section, but it does not gives optimistic bounds.

3. AFDX FLOW MODELLING ISSUES

In AFDX, all data flows are Virtual Links. A Virtual Link uses a static multicast path through the network and consists of a sporadic stream: frames of the stream are of bounded size, s^{max}, and there is a minimal duration between two successive frame transmissions (known as the "Bandwidth Allocation Gap", or BAG).

One can choose different arrival curves to model such a sporadic flow with maximal frame size $s^{\max}$ and inter-frame gap BAG. A first idea is to use a simple staircase function $\alpha(\Delta) = s^{max} \times \nu_{BAG,0}(\Delta) = s^{\max} \left\lceil \frac{\Delta}{BAG} \right\rceil$ (as presented in Figure 5). But one may prefer a more simple linear function $\alpha'(\Delta) = \gamma_{\frac{s^{max}}{BAG}, s^{max}}(\Delta) = s^{\max}(1 + \frac{\Delta}{BAG}) \wedge \delta_0$.

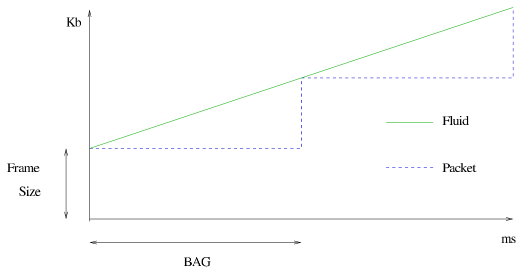

Figure 5: Fluid and discrete models of a sporadic flow

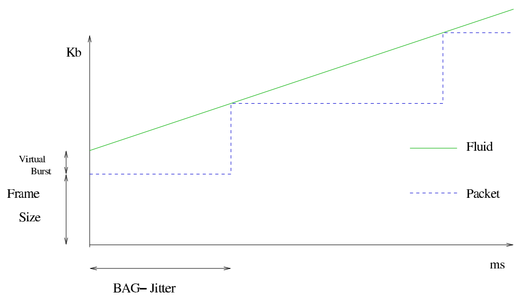

Figure 6: Fluid and discrete models of a sporadic flow with jitter

The staircase model is closer to the reality, but the computed delays in a single network element are, in common cases, the same with both modelling. The differences arise when crossing several nodes.

The fluid problem, highlighted in the introduction, arises when the flow crosses a sequence of servers. Assume that the first server is modelled by a jitter. In network calculus, this jitter leads to simply "shift" the curves (cf. Figure 6). But, in the fluid model, this shift creates a kind of virtual burst, that does not exist in the real system. In the second server, this virtual burst induces a pessimistic delay.

So, one may decide to use stair-case functions. But, the shaping cannot be (easily) modelled by stair-case functions. Shaping (or serialisation) basically means that, whatever the applicative flow, the output of a link cannot be greater than the link throughput. In network calculus, it means that the arrival curve of a flow sent on a link is the minimum between the link throughput (often a linear function) and the flow arrival curve [4]. Thus, to have an accurate modelling of both shaping and packets, both linear and stair-case functions are needed (Figure 7). It should be pointed out that there is a generic class of functions, called the Ultimate Pseudo Periodic function class (UPP, [3]), which generalises both stair-cases and CPL classes. This class has two drawbacks: first, it requires complex algorithms (and thus, a costly implementation), but above all, the basic operators on this class make intensive use of inversion and least common multiple (lcm), which implies an implementation with rationals of arbitrary size, excluding therefore a fast floating point based implementation.

4. AN ALGORITHM TO COMPUTE BOUNDS ON AFDX NETWORKS

We present in this section an algorithm to compute WCTT

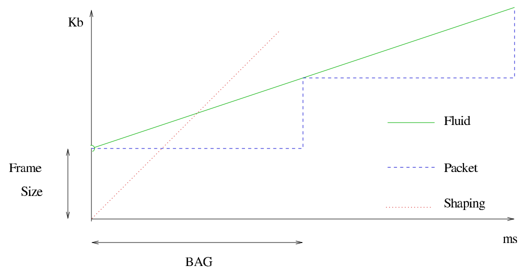

Figure 7: Shaping on fluid and discrete models of a sporadic flow

bounds on an AFDX network. In fact, it can be used for any FIFO store and forward networks, assuming that the basic operations on network calculus can actually be computed. This algorithm is parametrised by the arrival and service curves. We will then adapt this algorithm to make a combined use of two simple classes of functions, and provides more accurate results with a low computation time (Section 4.2), as it will be shown in the experiments of Section 5.

4.1 An algorithm for FIFO store and forward network

Let us consider a store and forward network, with FIFO policy, and without cyclic dependency. The network is made of switches interconnected by full duplex links. The modelling of a switch, illustrated in Figure 8, is as follows. Input ports are modelled as packetizers, the switching fabric consists in a single delay δ_d, with an important property: it keeps the flow packetized. Lastly, the output port is a server with a minimal curve and a shaping curve. A $n \times n$ switch is modelled in network calculus by $2n + 1$ servers: n packetizers, one delay, and n servers with minimal service and shaping curve. In the example of Figure 8, assuming a maximal size of 1Ko (*i.e.* 8Kb), a fabric delay of $16\mu s$ and output port running at 10Mb/s, servers S_1, S_2 are packetizers with $l^{max} = 8.10^3$, S_3 has a service curve δ_{16} and servers S_4, S_5 have service curve λ_{10} and also shaping curve λ_{10}.

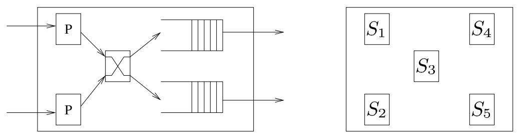

Figure 8: Switch model (2x2)

The algorithm applies on a set of server, $S_1, \ldots, S_n$, crossed by a set of end-to-end flows $\mathbf{F}_1, \ldots, \mathbf{F}_m$. Each end-to-end flow is itself a sequence of (local) flows (*i.e.* cumulative functions) and servers: $\mathbf{F}_i = F_i^0 \xrightarrow{S_{j_1}} F_i^1 \xrightarrow{S_{j_2}} \ldots \xrightarrow{S_{j_p}} F_i^p$ (the length of the end-to-end flow is denoted $|\mathbf{F}_i| = p$). One needs to encode the topology with two kinds of elements: flows and servers. We decide to use notations introduced for another kind of bipartite graph: Petri nets. The input (resp. output) flows of a server S_j is denoted ${}^\bullet S_j$ (resp. $S_j{}^\bullet$). Similarly, the source of a flow F_i^k is denoted ${}^\bullet F_i^k$ and its destination $F_i^{k\bullet}$. By induction, ${}^{\bullet\bullet}S_j$ is the set of nodes who have one output, that is an input of S_j, and $S_j{}^\bullet \cap {}^\bullet S_k$

is the (possibly empty) set of flows going from S_j to S_k[3].

The principle of the algorithm is the following. In each server, the global delay of the server is computed, and propagated on each flow using eq. (11). To compute this global delay, the arrival curve of the global input flow must be computed: it is not computed flow per flow, but grouping the flow per source. This is related to shaping: if 4 flows goes into a network element S (*i.e.* ${}^{\bullet}S = \{R_i\}_{i\in[1,4]}$), but they are in fact produced by only two nodes ($\{R_1, R_2\} = S'^{\bullet}$, $\{R_3, R_4\} = S''^{\bullet}$), considering that each flow R_i has the individual arrival curve α_i, *and* that the servers S', S'' have the shaping curves σ', σ'', the arrival curve for the global flow $R_1 + R_2 + R_3 + R_4$ is $(\alpha_1 + \alpha_2) \wedge \sigma' + (\alpha_3 + \alpha_4) \wedge \sigma''$.

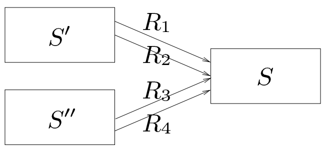

Figure 9: Grouping/shaping principle

Without loss of generality, to simplify the description algorithm, we assume that there is a specific source node S_0 such that all end-to-end flows are emitted by this source (*i.e.*, for all flow $\mathbf{F}_i$: ${}^{\bullet}(F_i^0) = S_0$). We also assume that each packetizer has only one source.

The algorithm requires some ordering for the computations: to compute the input of server S_j, one first needs to compute the output of all servers in ${}^{\bullet\bullet}S_j$. To avoid some topological sorting, the algorithm is written in a recursive way[4]. This leads to Algorithm 1: as an illustration, a partial computation is presented in Appendix A, and its proof is presented in Appendix B.

4.2 Juggling with two simple classes of service

Algorithm 1 involves some (min,+) operations on arrival curves (deconvolution, sub-additive closure, minimum, sum and horizontal deviation with service curve).

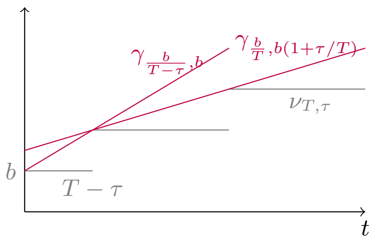

Figure 10: CPL overapproximation of a stair-case function

We consider simple linear shaping functions (*i.e.* $\sigma_{P_i} = \lambda_{R_i}$) and rate-latency services (*i.e.* $\beta_j = \beta_{R_j,T_j}$). If the arrival curves are linear $\gamma_{r,b}$ functions, *i.e.* a simple subset of CPL functions, all operations preserve the CPL shape of α_i^k functions.

Algorithm 1: Acyclic FIFO net – per server delay propagation and group shaping

```
   Input: Acyclic network description: servers
          $\{S_1,\ldots,S_n\}$, paths of flows $(\{\mathbf{F}_1,\ldots,\mathbf{F}_m\})$
   Input: Arrival curves of initial flows $(F_i^0 \prec \alpha_i)$; Curves
          of minimal service $(S_j \trianglerighteq \beta_j)$ ; Shaping curves
          $\sigma_j$ of each $S_j$
   Data: Computed arrival curves of flows: $\alpha_i^k$; Total
         input for server $S_j$: $\alpha^{S_j}$;
   Result: Maximum delay for each server: $d_j$; Maximum
           delay for each flow: $d_i^F$
 1 Function $GroupAC(S_j, \mathcal{F}\ )$ ;
 2 Assert($\mathcal{F} \subset S_j{}^{\bullet}$) ;
 3 begin
 4     if $S_j = S_0$ then
 5         return $\sum_{F_i^k\in\mathcal{F}} \alpha_i^k$ ;        // $\mathcal{F} \subset S_j{}^{\bullet} \implies k = 0$
 6     else if $S_j$ is an aggregate server then
 7         if $d_j = null$ then
 8             $\alpha^{S_j} \leftarrow \sum_{S_p\in{}^{\bullet\bullet}S_j}$ GroupAC($S_p, S_p{}^{\bullet} \cap {}^{\bullet}S_j$) ;
 9             $d_j = h(\alpha^{S_j}, \beta_j)$;
10             foreach $F_i^k \in {}^{\bullet}S_j$ do
11                 $\alpha_i^{k+1} \leftarrow \left(\alpha_i^k \oslash \delta_{d_j}\right)^*$ ;
12         return $\left(\sigma_j \wedge \sum_{F_i^k\in\mathcal{F}} \alpha_i^k\right)^*$
13     else if $S_j$ is a P-packetizer with per flow maximal
       packet size $l_i^{max}$ then
14         Assert($|{}^{\bullet\bullet}S_j| = 1$) ;
15         if $d_j = null$ then
16             $d_j \leftarrow 0$ ;
17             GroupAC(${}^{\bullet\bullet}S_j, \emptyset$) ;
18             foreach $F_i^k \in {}^{\bullet}S_j$ do
19                 $\alpha_i^{k+1} \leftarrow \alpha_i^k$
20         $l_g^{max} = \bigvee_{F_i^k\in\mathcal{F}} l_i^{max}$ ;
21         $S_P \leftarrow {}^{\bullet\bullet}S_j$ ;
22         return $\left((\sigma_P + l_g^{max}) \wedge \sum_{F_i^k\in\mathcal{F}} \alpha_i^k\right)^*$
   // Main
23 begin
       // Initialisations
24     for $j = 1$ to $n$ do
25         $d_j \leftarrow null$ ;
26     for $i = 1$ to $m$ do
27         $\alpha_i^0 \leftarrow \alpha_i{}^*$ ;
       // Forcing evaluation for each server
28     for $j = n$ downto 1 do
29         GroupAC($S_j, \emptyset$)
       // Per flow delay
30     for $i = 1$ to $m$ do
31         $d_i^F \leftarrow \sum_{k=1}^{|\mathbf{F}_i|} d_{j_k}$ ;
           // with $\mathbf{F}_i = F_i^0 \xrightarrow{S_{j_1}} \cdots \xrightarrow{S_{j_n}} F_i^n$
```

[3]An example is provided in Appendix A

[4]with some test to avoid to redo twice the same computations

Method	CPL (float)	CPL/$b.\nu_{T,\tau}$ (float)	UPP (rat)
Computation time	0.9 s	1.1 s	7.2 s
Min gain	-	0%	0.15%
Max gain	-	7.8%	15.2%
Av. gain	-	2.49%	5.92%
Min gain on 1000 biggest	-	0.8%	2.0%
Max gain on 1000 biggest	-	4.4%	11.9%
Av. gain on 1000 biggest	-	2.9%	8.3%

Table 1: Comparison of the 3 methods on the case-study

If the arrival curves are stair-case functions $b \cdot \nu_{T,\tau}$, then the lines 11 and 22 lead out of the stair-cases class[5].

As discussed in Section 3, the UPP class of function is able to handle all of these operations but it has some severe performance limitations. Our proposal is to modify lines 11 and 22. Consider a stair-case function $b \cdot \nu_{T,\tau}$. As shown in Figure 10, it can be over-approximated by a two-slope CPL function.

$$b\nu_{T,\tau} \leq \gamma_{\frac{b}{nT-\tau},nb} \wedge \gamma_{\frac{b}{T},b(1+\tau/T)} = cpl_{b,T,\tau} \tag{14}$$

with $n = \lfloor \frac{\tau}{T} \rfloor + 1$ and $\lfloor x \rfloor$ the floor function: $\lfloor x \rfloor \in \mathbb{N}, x-1 < \lfloor x \rfloor \leq x$).

Let us denote $cpl(b \cdot \nu_{T,\tau})$ the function associating the CPL over-approximation of a stair-case function. Then, if $b\nu_{T,\tau}$ is an arrival curve for a flow R, also is $cpl(b\nu_{T,\tau})$. The idea of the algorithm adaptation is to replace sum of α_i^k by sum of $cpl(\alpha_i^k)$. Explicitly, lines 12 and 22 are respectively replaced by

$$\textbf{return } \sigma_j \wedge \sum_{F_i^k \in \mathcal{F}} cpl(\alpha_i^k)$$

$$\textbf{return } (\sigma_P + l_g^{max}) \wedge \delta_0 \wedge \sum_{F_i^k \in \mathcal{F}} cpl(\alpha_i^k)$$

Proof. We have to show that the new expressions remain arrival curves of $\mathcal{F}$. This is done by over-approximation (cf eq 5).

Since, $\left(\sigma_j \wedge \sum_{F_i^k \in \mathcal{F}} \alpha_i^k\right)^*$ is an arrival curve for $\mathcal{F}$, also is $\sigma_j \wedge \sum_{F_i^k \in \mathcal{F}} \alpha_i^k$ ( ) and then $\sigma_j \wedge \sum_{F_i^k \in \mathcal{F}} cpl(\alpha_i^k)$. Moreover, since the CPL class is closed by minimum and sum, the expression is a CPL function. The use of the sub-additive closure is useless since CPL functions are sub-additive.

The same reasoning applies for $(\sigma_P + l_g^{max}) \wedge \sum_{F_i^k \in \mathcal{F}} cpl(\alpha_i^k)$. The minimum with δ_0 is just a sub-additive closure computation: in σ_P is a CPL function, $\sigma_P + l_g^{max}$ also is, up to a problem value at 0. The minimum with δ_0 put the expression back into the CPL class. □

Finally, line 11 can be replaced by $\alpha_i^{k+1} \leftarrow (\alpha_i^k \oslash \delta_{d_j}) \wedge \delta_0$, since the two expressions are equivalent when α_i^k is a stair-case function. In the same way, line 27 can be replaced by $\alpha_i^0 \leftarrow \alpha_i$.

5. EXPERIMENTS

The experiments are conducted on a realistic industrial configuration provided by Thales Avionics, with 104 nodes connected to an AFDX network having 8 switches. The number of virtual links amounts to 974 leading to 6501 WCTT latency constraints. This case-study has been first introduced in [5].

The configuration is analysed with 3 methods implemented in the RTaW-Pegase WCTT evaluation tool [5]. The first method, denoted by CPL in Table 11, uses Algorithm 1 with the fluid modelling of arrival curve where each VL i has an arrival curve $\alpha_i' = \gamma_{\frac{s_i^{max}}{BAG_i}, s_i^{max}}$. Then, the computation is done within the CPL function class, using floating point numbers. The second method, denoted UPP in Table 11, also uses Algorithm 1 but with stair-case arrival curves: $\alpha_i = s_i^{max} \times \nu_{BAG_i,0}$. Then, the delay propagation requires the general UPP function class, which implies rational computation. The third approach , denoted by CPL/ν in Table 11, implements the algorithm adaptation presented in Section 4.2, also considering stair-case arrival curves, but the computation remains in the realm of floating point numbers.

The results of the experiments are presented in Table 1, Figures 11 and Figures 12, the latter being a zoom on the results for the 1000 paths with the largest delays. When considering that the reference method is the fluid modelling, the new algorithm provides a 2.9% improvement, at the expense of an increase of 22% of the computation time, whereas the UPP modelling provides an 5.9% improvement, at the expense of an increase of 700% of the computation time.

These results suggest that the approach developed in this paper offers a reasonable trade-off between the two other methods. The computation time is 22% larger than the simple CPL method, and 84% smaller than the UPP one. The accuracy of the results also is between both other methods, as shown in Figures 11. When the UPP gain is on average equal to 5.92%, CPL/ν leads to an average gain of 2.9%.

6. CONCLUSION

Network calculus has been used for more than 15 years to compute guaranteed upper bounds in networks. However, in large scale systems such as avionics ones, computation times can be an issue. To make network calculus more practical and suited to a wider range of use-cases, algorithms corresponding to different trade-offs between computation time and result accuracy should be developed.

This paper considers the case of a FIFO network, and makes some adaptations to a common algorithm in order to obtain a meaningful trade-off between existing solutions. This proposal can be of particular interest for design space exploration. Experiments conducted on a realistic AFDX configuration confirm the initial intuition: when considering that the reference method is the fluid modelling, the new algorithm provides a 2.9% improvement, at the expense of an increase of 22% of the computation time, whereas the most detailed modelling provides a 5.9% improvement at the expense of an increase of 700% of the computation time.

Another contribution of the paper is that it provides, to the best of our knowledge, the first detailed presentation and proof of the delay computation algorithm in the FIFO case.

7. REFERENCES

[5] The sum of stair-cases is not a simple stair-case function, neither is the minimum with a linear function, nor the sub-additive closure.

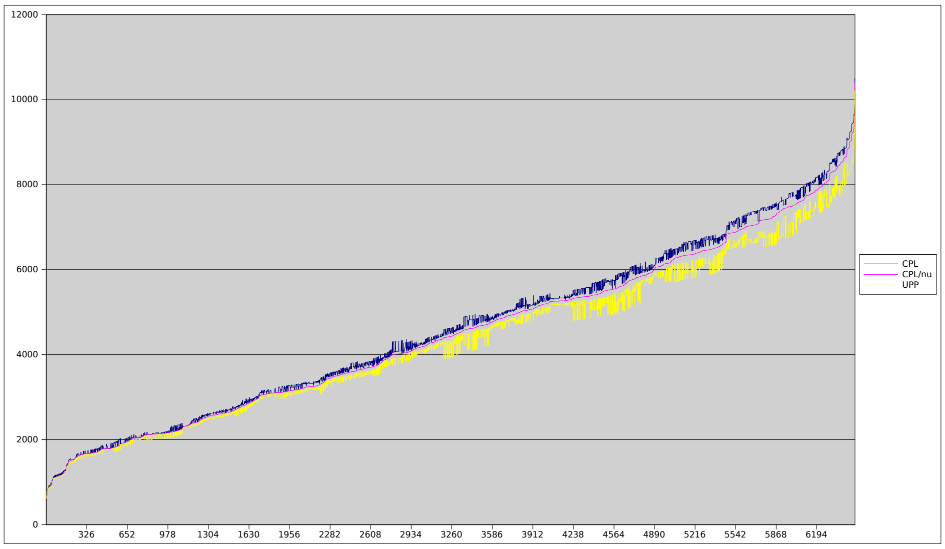

Figure 11: Path delay bounds with the 3 methods (VLs are sorted by increasing delay computed with the second method)

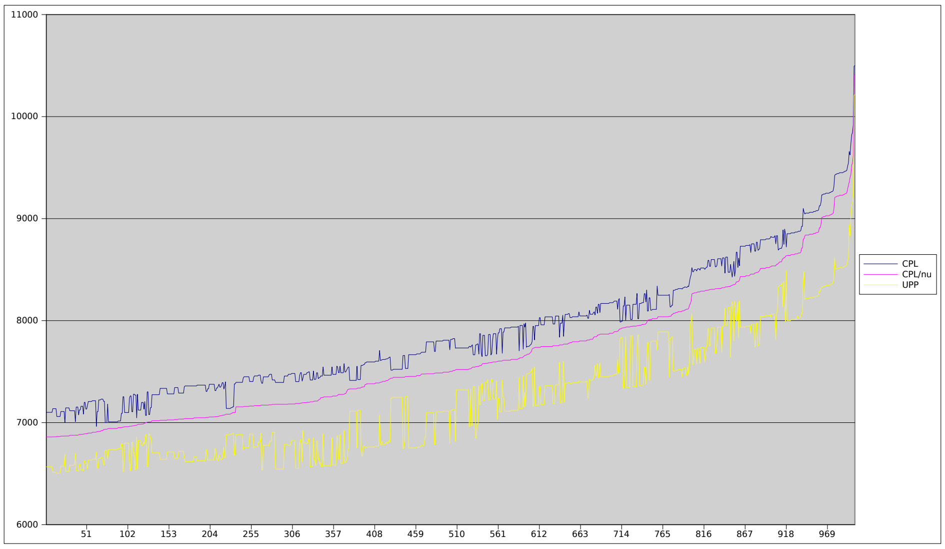

Figure 12: Zoom on the last part of Figure 11

[1] Airlines Electronics Eng. Committee (AEEC). ARINC 664: Aircraft data network, part 7: Avionics full-duplex switched ethernet (AFDX) network, September 2009.

[2] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Worst-case end-to-end delay analysis of an avionics AFDX network. In *Proc. of the Design, Automation and Test in Europe conference (DATE 2010)*, Dresden, Germany, 8–12 March 2010.

[3] A. Bouillard and E. Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 17(4), october 2007.

[4] M. Boyer. Half-modeling of shaping in FIFO net with network calculus. In *Proc. of the 18th International Conference on Real-Time and Network Systems (RTNS 2010)*, Toulouse, France, November 4-5 2010.

[5] M. Boyer, J. Migge, and M. Fumey. PEGASE, a robust and efficient tool for worst case network traversal time. In *Proc. of the SAE 2011 AeroTech Congress & Exhibition*, Toulouse, France, 2011. Preliminary version available at `http://www.realtimeatwork.com`.

[6] C.-S. Chang. *Performance Guarantees in communication networks*. Telecommunication Networks and Computer Systems. Springer, 2000.

[7] COINC home page. http://www.istia.univ-angers.fr/~lagrange/software.php. Retrieved on 12 September 2011.

[8] R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on information theory*, 37(1):114–131, January 1991.

[9] R. L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on information theory*, 37(1):132–141, January 1991.

[10] F. Frances, C. Fraboul, and J. Grieu. Using network calculus to optimize AFDX network. In *Proceeding of the 3thd European congress on Embedded Real Time Software (ERTS06)*, Toulouse, January 2006.

[11] J. Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques.* PhD thesis, Institut National Polytechnique de Toulouse (INPT), Toulouse, Juin 2004.

[12] J.-Y. Le Boudec and P. Thiran. *Network Calculus*, volume 2050 of *LNCS*. Springer Verlag, 2001.

[13] X. Li, J.-L. Scharbarg, and C. Fraboul. Improving end-to-end delay upper bounds on an AFDX network by integrating offsets in worst-case analysis. In *Proc. of the 15th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2010)*, pages 1–8, Bilbao, Spain, September 2010.

APPENDIX

A. EXAMPLE FOR ALGORITHM 1

The Algorithm 1 is illustrated with a simple example[6] where three flows are produced by two end-systems, crossing one 2x2 switch (Figures 13). The system is modelled with three flows $\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3$ and six servers $S_1, \ldots, S_6$. The servers S_3, S_4 are packetizers, and the others are aggregate servers. To give an overview of notations: $S_2{}^\bullet = \left\{\mathbf{F}_2^1, \mathbf{F}_3^1\right\}$, ${}^\bullet S_5 = \left\{F_1^2, F_2^2\right\}$, ${}^\bullet F_1^2 = S_3$, ${}^{\bullet\bullet} S_5 = \{S_3, S_4\}$, $S_4{}^\bullet \cap {}^\bullet S_5 = \left\{\mathbf{F}_2^2\right\}$.

First of all, the arrival curves are initialised ($\alpha_i^0 \leftarrow \alpha_i{}^*$).

The algorithm first computes $\texttt{GroupAC(}S_6, \emptyset\texttt{)}$. Since this is the first times this servers is crossed ($d_6 = \mathit{null}$), α^{S_6} is computed as $\texttt{GroupAC(}S_4, \mathbf{F}_3^2\texttt{)}$. An anthropomorphic analogy would be: "Hey, S_4, give me an arrival curve for $\mathbf{F}_3^2$".

Now, control is passed to S_4, that requires the computation of $\texttt{GroupAC(}S_4, \emptyset\texttt{)}$, which leads to the computation of $\alpha^{S_2} = \texttt{GroupAC(}S_0, \left\{\mathbf{F}_2^0, \mathbf{F}_3^0\right\}\texttt{)}$.

The control has reached the origin S_0, and the result is simply the sum of the arrival curves: $\texttt{GroupAC(}S_0, \left\{\mathbf{F}_2^0, \mathbf{F}_3^0\right\}\texttt{)} \leftarrow \alpha_2^0 + \alpha_3^0 = \alpha_2{}^* + \alpha_3{}^*$ (cf eq 8).

With this information, S_2 can computes its own delay, d_2, and propagate it to $\mathbf{F}^1_{i \in \{1,2\}}$): $\alpha_i^1 = \alpha_i^0 \oslash \delta_{d_2}$ (eq. 11).

The packetizer S_4 now, first computes the arrival curve of its output (*i.e.* $\alpha_3^2 = \alpha_3^1$, by application of eq. 11 and eq. 12), and secondly can give $\texttt{GroupAC(}S_4, \mathbf{F}_3^2\texttt{)}$ as the minimum of $\mathbf{F}_3^2$ arrival curve and its own shaping curve ($\sigma_2 + l_4^{max}$).

At last, S_6 is able to computes its own delay $d_6 = h((\sigma_2 + l_4^{max}) \wedge \alpha_3^2, \beta_6)$. And the end-to-end delay of $\mathbf{F}_3$ is $d_3^F = d_2 + d_6$.

The same can be done from S_5 for flows $\mathbf{F}_1$ and $\mathbf{F}_2$.

B. PROOF OF ALGORITHM 1

The main function is $\texttt{GroupAC(}S_j, \mathcal{F}\texttt{)}$. The semantics of this function is to compute an arrival curve for the group of flows $\mathcal{F}$ (*i.e.* $\mathcal{F} \prec \texttt{GroupAC(}S_j, \mathcal{F}\texttt{)}$), assumed that all flows in $\mathcal{F}$ have the same source S_j (the parameter S_j could have been deduced from $\mathcal{F}$ but it is given to improve readability).

[6] To keep it simple, the switch fabric is omitted in the modelling and also are the sub-additive closures in the algorithm.

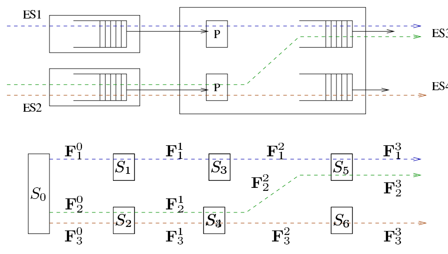

Figure 13: Topology example

This is not a pure-functional programming, since this function updates some global variables (d_j, α_i^k).

So, we have to prove that, for all set $\mathcal{F}$, assuming that ${}^\bullet\mathcal{F} = \{S_j\}$, $\mathcal{F} \prec \texttt{GroupAC(}S_j, \mathcal{F}\texttt{)}$.

First case (lines 4–5): if the source is the special S_0 source, then just makes the sum of the arrival curves (initialised at line 27).

Second case (lines 6–12): the flows are output out of a FIFO aggregate server. Then, the result eq. (11) can be used to consider only the delay introduced by S_j.

Then, one first need to compute this delay (lines 7–11). The test in line 7 is just an optimisation to makes this computation only once. The interesting part of this algorithm is at line 8: instead of making the sum of all $\alpha_i^k \in {}^\bullet S_j$, this set is partitioned by input (${}^\bullet S_j = \bigcup_{S_P \in {}^{\bullet\bullet} S_j} S_p{}^\bullet \cap {}^\bullet S_j$, then, $\sum_{F_i^k \in {}^\bullet S_j} F_i^k = \sum_{S_P \in {}^{\bullet\bullet} S_j} \sum_{F_i^k \in S_p{}^\bullet \cap {}^\bullet S_j} F_i^k$. Then, by induction, if $\texttt{GroupAC(}S_p, S_p{}^\bullet \cap {}^\bullet S_j\texttt{)}$ is an arrival curve for $S_p{}^\bullet \cap {}^\bullet S_j$, the per S_P sum (α^{S_j}) is an arrival curve for ${}^\bullet S_j$.

This delay can be propagated to all input flows (using eq. (11) and tighten with the sub-additive closure eq. (4)) (line 11). Notice that we don't use the shaping here ($\alpha_i^k \wedge \sigma_j$ is also an arrival curve for F_i^k, but it is not used there, because of packetizer).

Line 12 computes $\texttt{GroupAC(}S_j, \mathcal{F}\texttt{)}$: since α_i^k is an arrival curve for each F_i^k, $\sum_{F_i^k \in \mathcal{F}} \alpha_i^k$ is an arrival curve for $\mathcal{F}$. But S_j is a shaper, then σ_j is an arrival curve for $S_j{}^\bullet$, and, since $\mathcal{F} \subset S_j{}^\bullet$, σ_j also is an arrival curve for the subset $\mathcal{F}$ (eq. (9)), and one can get the minimum of both (eq. (6)).

Third case (lines 13–22): the flows are the output of a packetizer. Using (eq. (12)), the packetizer does not introduce any delay ($d_j \leftarrow 0$), and the output arrival curves can be computed based on the delay. If S_P is the previous node, $\forall F_i^{k'} \in {}^\bullet S_P : d(F_i^{k'}, S_P; S_j) = d(F_i^{k'}, S_P)$, and $\alpha_i^{k'+2} = \alpha_i^{k'} \oslash \delta_{d(F_i^{k'}, S_P)}$. Now, consider $k' = k - 1$, and, from line 11, $\alpha_i^k = \alpha_i^{k'} \oslash \delta_{d_P}$ and $d_P \geq d(F_i^{k'}, S_P)$. Then, $\alpha_i^k = \alpha_i^{k+1}$ is an arrival curve for F_i^{k+1}.

Then, at last, the shaping is modelled: using eq. (13). Since σ_P is the shaping curve of S_P, and ${}^\bullet S_j \subset S_P{}^\bullet$, ${}^\bullet S_j \prec \sigma_P$, then $\sigma_P + l_g^{max}$ is an arrival curve for all outputs of S_j, and, in particular, for $\mathcal{F}$. And, as in line 12, $\sum_{F_i^k \in \mathcal{F}} \alpha_i^k$ also is.

One can conclude that:

$$\mathcal{F} \prec \texttt{GroupAC(}S_j, \mathcal{F}\texttt{)} \tag{15}$$

The loop at line 28 is just a way to ensure that all d_j are computed, and the one at line 30 computes, for each flow, its end-to-end delay as the sum of the local delays.